Doc Ref. AN1
Appl. No. 09/892,482

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 061 458 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
20.12.2000 Bulletin 2000/51

(51) Int. Cl.^{7}: G06F 17/30, G06F 12/08

(21) Application number: 00304763.6

(22) Date of filing: 06.06.2000

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: 15.06.1999 US 334010

(71) Applicant:
SUN MICROSYSTEMS, INC.
Palo Alto, California 94303 (US)

(72) Inventors:
- Graham, Neil
Nepean, Ontario K2H 8A9 (CA)
- Leduc, Kevin
Ottawa, Ontario KIN-7S4 (CA)

(74) Representative:
Harris, Ian Richard et al
D. Young & Co.,
21 New Fetter Lane
London EC4A 1DA (GB)

(54) Caching of reduced forms of web pages on a small footprint device

(57) A system and method to cache reduced forms of web pages for use in a web browser. A web browser running on a small footprint device may cache reduced forms of web pages. Various types of reduction may be performed. For example, web pages may comprise elements which are unnecessary to display or are unsupported for particular small footprint devices. These elements may be removed before a web page is cached, which may reduce the size taken in the cache and reduce the time to render the page for a subsequent view. Also, a version of the parse tree may be cached instead of the web page in text form. A lightweight containment framework for applications and services running on small footprint devices is described. A web browser operable to cache reduced forms of web pages for small footprint devices may be built on this containment framework.

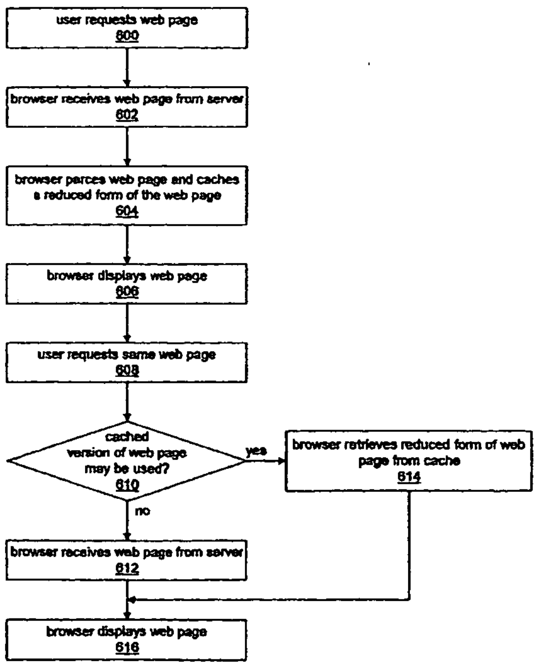
user requests web page 600
browser receives web page from server 602
browser parces web page and caches a reduced form of the web page 604
browser displays web page 606
user requests same web page 608
cached version of web page may be used? 610
yes
no
browser retrieves reduced form of web page from cache 614
browser receives web page from server 612
browser displays web page 616

Fig. 1

EP 1 061 458 A2

Description

BACKGROUND OF THE INVENTION

1. Field of the Invention

[0001] This invention relates generally to computer application programs and small footprint devices. More particularly, the invention comprises a system and method for caching reduced forms of web pages for display in a web browser running on a small footprint device.

2. Description of the Relevant Art

[0002] The field of "smart" small footprint devices is growing and changing rapidly. Small footprint devices include handheld computers, personal data assistants (PDAs), cellular phones, global positioning system (GPS) receivers, game consoles, and many more such devices. These devices are becoming more intelligent and interconnected. Technologies such as Jini™ from Sun Microsystems, Inc. and initiatives such as the Open Service Gateway Initiative (OSGI) are expanding the traditional concepts of computer networks to include small footprint devices.

[0003] This increased device interconnection has introduced a need for both new types of computing services and new ways to integrate computing services, both inter-device-based and intra-device-based services. A "service" is an entity implemented within or accessible from a device that can be used by a person, an application, or another service. The concept of a service is broad and can be considered at many different scales. For example, services include familiar network-based services such as shared printing, email, telephony, etc. Services also include less familiar examples such as an energy management service which may control the power consumption of devices within a local network, a diagnostic service which allows a device to send information to a service technician when an error occurs, a health-monitoring service which immediately notifies health professionals of an emergency, etc.

[0004] Services also include modules or applications located and executable within a local machine or device. For example, local application programs may utilize a service to communicate with an HTTP server, an HTML render engine service, a bookmark service, a user interface service, etc. In this example, an application program may use these services together to implement a web browser program.

[0005] It is becoming more common today to execute multiple services and applications together in a single small footprint device. However, since memory, processing power, and other resources are typically very limited in small footprint devices, a specialized lightweight service/application containment framework is necessary to achieve the desired integration of services and applications. It is also desirable that the containment framework be flexible and extendable enough to provide a framework for any types of services and applications for any kind of small footprint device. A further goal may be that the containment framework be compatible and integrated with off-device services such as services available to devices in a Jini™ network. The containment framework described herein achieves the above-stated goals.

[0006] The lightweight containment framework may enable small footprint devices such as personal data assistants, smart cellular phones, etc. to run the types of multi-purpose application programs traditionally associated with desktop computing environments. For example, the Personal Applications Suite available from Sun Microsystems, Inc. is built around one embodiment of the containment framework. The Personal Applications suite comprises an integrated set of compact, memory-efficient applications, including the Personal Applications Browser, the Personal Applications Email Client, and the Personal Organizer.

[0007] Since small footprint devices may have very strong resource constraints, it is important that not only the application and service framework be lightweight, but also that the applications and services themselves be as compact and efficient as possible. Given the utility and popularity of the world wide web and the movement toward "smart" small footprint devices, web browsers in particular are one type of application program where compactness, speed, and efficiency are large concerns. The present invention comprises a system and method to cache reduced forms of web pages for use in a web browser. A web browser is a program allowing users to view web pages.

SUMMARY OF THE INVENTION

[0008] Particular and preferred aspects of the invention are set out in the accompanying independent and dependent claims. Features of the dependent claims may be combined with those of the independent claims as appropriate and in combinations other than those explicitly set Out in the claims.

[0009] The present invention comprises a system and method to cache reduced forms of web pages for use in a web browser. Web pages may comprise HTML code or other markup language code, such as XML-derived markup language code, scripting code, style sheets, etc. There are several reasons why it may be desirable for a web browser running on a small footprint device to cache a reduced form of a web page, rather than simply caching an unprocessed version of the web page. For example, a typical small footprint device may have strong memory constraints compared to other computing systems such as a desktop computing system. Thus, it may be important to reduce memory usage for web page caching by compacting the cached content in some way. Also, a small footprint device may have limited processing power compared to other computing systems. Thus it may be desirable for a small footprint device web browser to save CPU cycles by caching partially or completely parsed web pages in order to avoid having to reparse a web page.

[0010] Two basic approaches to caching reduced forms of web pages are described herein. In one approach, the browser may cache data comprising markup language code, style sheets, etc. but superfluous, unnecessary, or inapplicable data, such as certain markup language tags, may be removed. For example, many web pages comprise HTML or scripting code for multimedia or visual effects that cannot be displayed on certain small footprint devices, or can be removed in the interest of saving system resources.

[0011] In another approach, the browser may cache a parsed form of a web page rather than the raw web page. For example, a web page may be parsed into a tree, such as a tree of objects according to the Document Object Model standard, and this tree may be stored in binary form in a cache.

[0012] A lightweight containment framework for applications and services running on small footprint devices is described herein. Such a containment framework enables applications/services to be constructed using a modular architecture. For example, a web browser application may utilize a parser service module, a caching service module, a user interface service module, etc. It may be desirable to implement the caching of reduced web pages described herein in a web browser built using such a modular architecture. For example, a web browser may use a parsing service to produce a parse tree, and the parsing service may use a storage service which caches the parse tree, using whatever means are appropriate for a particular device or system.

[0013] As another example, a web browser may use a parsing service to parse a web page, and the parsing service may use another service to remove unnecessary data, such as unsupported HTML tags, from the web page. A modular software architecture enables a web browser to remove different types of data from web pages for different types of devices.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] Other objects and advantages of the invention will become apparent upon reading the following detailed description and upon reference to the accompanying drawings in which:

Figure 1 is a flowchart diagram illustrating the process of caching and reusing a reduced form of a web page;

Figure 2 is a block diagram illustrating the hardware architecture of a typical small footprint device;

Figure 3 illustrates a typical hierarchy of hardware/software layers involved in a system running applications and services within the containment framework;

Figure 4 illustrates an exemplary network in which a small footprint device running applications/services in the containment framework is connected to a local service-based network;

Figure 5 illustrates the discovery process, in which a service provider finds a lookup service;

Figure 6 illustrates the join process, in which a service provider registers its service with a lookup service;

Figure 7 illustrates the lookup process, in which a client requests a service from a lookup service;

Figure 8 illustrates the service invocation process, in which a client invokes a service using a service object received from a lookup service;

Figure 9 is an abstract block diagram illustrating the basic architecture of the containment framework;

Figures 10 and 11 illustrate the use of module request listeners in the containment framework to simulate a hierarchical containment environment;

Figure 12 illustrates the use of parcels to group modules together;

Figure 13 is a flowchart diagram illustrating a typical lookup process that the central framework instance may perform when it receives a lookup request for a service module from a client module; and

Figure 14 is a flowchart diagram illustrating the module release process.

[0015] While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and are herein described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the scope of the present invention as defined by the appended claims.

DETAILED DESCRIPTION OF THE INVENTION

Figure 1 - Caching a Reduced Web Page

[0016] Figure 1 is a flowchart diagram illustrating the process of caching and later reusing a reduced form of a web page. It is noted that Figure 1 illustrates one embodiment of the process. Other embodiments are possible in which various steps are added, combined, modified, or omitted.

[0017] In step 600 of Figure 1, a user requests a web page. This request may occur in various ways. For example the user may click on a web link, or the user may enter a URL in the browser, or the user may press a button to refresh the current page, etc. Also, the page may not be requested by a user per se. For example, another application may request the web page in step 600. In step 602 a web browser communicates with a server to receive the requested web page. For example, the browser and server may utilize the hypertext transfer protocol (HTTP) to transfer a web page from the server to the web browser. The web page may contain various types of data such as text, HTML code or other markup language code, scripting code such as Javascript™ code, style sheet directives, etc.

[0018] In step 604, the web browser parses the web page and caches a reduced form of the web page. The browser may use any of various types of caching mechanisms. The cache may be a portion of RAM or other type of memory, as appropriate for a particular system. In one embodiment, the browser may use a caching service to store the reduced web page.

[0019] As described above, various types of web page reductions may be used in step 604. In one embodiment, the browser parses the web page to produce a parse tree according to the Document Object Model (DOM) standard. The Document Object Model is an application programming interface (API) for web pages. It defines the logical structure of documents and the way a document is accessed and manipulated. A web browser may use the parse tree to lay out the web content for display.

[0020] As described above, small footprint devices may have very limited resources such as memory and processor capacity. In contrast to other types of computing systems in which the time and system resources required to parse a web page may be negligible, this parsing may require significant portions of system resources in a small footprint device. Thus, the web browser may cache the parse tree for a web page after the page has been parsed a first time, and the browser may use the cached parse tree to display the web page in response to subsequent requests for the page. For example, the browser may serialize the parse tree into a memory image and cache the memory image.

[0021] The web browser may also reduce a web page for caching in other ways. These other forms of reducing web pages may be used together with or as an alternative to the technique of caching a parse tree. In one embodiment, the browser may remove particular elements from the web page as the page is parsed. The elements removed comprise elements such as HTML tags or XML-derived markup language tags, style sheet attributes, etc., which are unsupported for the particular small footprint device or are unnecessary. For example, web pages often include code to produce visual effects such as blinking text, etc. which may be unsupported or may unnecessarily consume memory and processor resources in a small footprint device. Web pages may also include dynamic scripting code which is unnecessary or unsupported in a small footprint device, such as mouseover effects, animation, etc. By removing these types of elements, a reduced form of the web page is produced which may take less space to cache and may be quicker to parse than the original page.

[0022] In some cases, the web browser may substitute other elements in the web page which make the page quicker to parse and render and/or smaller to display. For example, a web browser may replace an HTML table by simply listing the content of the cells sequentially, making it unnecessary to parse a table to determine column widths, row heights, etc.

[0023] In step 606 the browser displays the web page. As described above, the browser may parse the web page into a tree in step 604 and then use this tree to lay out and display the web page in step 606.

[0024] In step 608 the user requests the same web page as the web page requested in step 600. Step 606 and 608 may not be sequential steps. For example, a user may navigate through several other web pages before again requesting the web page of step 600 in step 608. As in step 600, the request for the web page may take various forms.

[0025] In step 610, the web browser determines whether the cached version of the web page may be used. The browser may employ various types of caching algorithms for web pages, which are well known in the art, to make this determination. If the web page must be received again from the server, then in step 612 the web browser communicates with the server to receive the web page. However, if the cached version of the page may be used, then in step 614, the web browser retrieves the reduced form of the web page that was stored in the cache in step 604. The browser may retrieve the reduced version of the web page in various ways, depending on what form the page was stored in, in step 604. For example, if the page was stored as a serialized parse tree, then the parse tree may be read into memory. Although the retrieval process of step 614 may consume device resources, the overall expense of displaying the page may be less than if the page must be received again and reparsed, as in step 612.

[0026] In step 616, the web browser displays the page received in either step 612 or step 614.

<u>Small Footprint Device Application/Service Containment Framework</u>

[0027] A modular application/service framework may be necessary or desirable to implement the type of web browser described herein which caches reduced forms of web pages. The containment framework described below allows for applications/services running on a small footprint device to be built from various modules. For example, this containment framework may allow the parsing and caching services to be implemented in a separate service modules. Appropriate service modules may then be used for different small footprint devices.

Figure 2 - Hardware Architecture Block Diagram

[0028] Figure 2 is a block diagram illustrating the hardware architecture of a typical small footprint device. As used herein, a small footprint device is a hardware device comprising computing resources such as a processor and a system memory, but having significantly greater constraints on one or more of these resources than a typical desktop computer has. For example, a small footprint device may have two megabytes of memory or less, whereas a typical desktop system may have 64 megabytes or more. Also a typical small footprint device may have significantly less processing power than a typical desktop computing system, either in terms of processor type, or processor speed, or both. For example, a personal data assistant device may have a 16 MHz processor, whereas a typical desktop system may have a processor speed of 100 MHz or higher. Also, a typical small footprint device may have a display size significantly smaller than the display screen of a desktop computing system. For example, the display screen of a handheld computer is typically small compared to the display screen of a desktop monitor.

[0029] It is noted that the specific numbers given are exemplary only and are used for comparison purposes. For example, a personal data assistant having eight megabytes of memory or more may still be a small footprint device, although the device has more memory than the typical figure of two megabytes given above.

[0030] Small footprint devices may also have constraints on other resource types compared to typical desktop computing systems, besides the memory, processor, and display size resources described above. For example, a typical small footprint device may not have a hard disk, may not have a network connection, or may have an intermittent network connection, or may have a wireless network connection, etc.

[0031] Many small footprint devices are portable and/or are small compared to desktop computers, but are not necessarily so. Also, many small footprint devices are primarily or exclusively battery-operated. Also, small footprint devices may typically have a more limited or narrow range of usage possibilities than a typical desktop computing system. Small footprint devices include, but are not limited to, the following examples: handheld computers, wearable devices (e.g., wristwatch computers), personal data assistants (PDAs), "smart" cellular telephones, set-top boxes, game consoles, global positioning system (GPS) units, electronic textbook devices, etc. Since new classes of consumer devices are rapidly emerging, it is not possible to provide an exhaustive list of small footprint devices. However, the term "small footprint device" is intended to include such devices as may reasonably be included within the spirit and scope of the term as described above.

[0032] Figure 2 illustrates a block diagram of a typical small footprint device. It is noted that the small footprint device may have various different architectures, as desired. The hardware elements not necessary to understand the operation of the present invention have been omitted for simplicity.

[0033] As shown in Figure 2, the small footprint device contains a processor 100. The processor 100 may be any of various types, including an x86 processor, e.g., a Pentium class, a PowerPC processor, as well as other less powerful processors or processors developed specifically for small footprint devices. The processor 100 may have various clock speeds, including clock speeds similar to those found in desktop computer-class processors, as well as lower speeds such as 16 MHz.

[0034] Also shown in Figure 2 the device includes a system memory 102. The system memory 102 may comprise memory of various types including RAM or ROM. A typical small footprint device may have a very small memory storage capacity compared to a typical desktop computer system.

[0035] A small footprint device may also comprise one or more input mechanisms. An input mechanism 104 is illustrated in Figure 2. The input mechanism 104 may be any of various types, as appropriate to a particular device. For example, the input mechanism may be a keypad, mouse, trackball, touch pen, microphone, etc.

[0036] A small footprint device may also comprise one or more display mechanisms. A display 106 is illustrated in Figure 2. However, a small footprint device may not comprise a display, or may comprise another type of output mechanism, such as an audio speaker. The display mechanism 106 may be any of various types, as appropriate to a particular device. The display mechanism for a typical small footprint device, such as a smart cellular phone, may be small compared to the display of a desktop computer system.

Figure 3 - Hardware/Software Hierarchy Diagram

[0037] Figure 3 illustrates a typical hierarchy of hardware/software layers involved in a system running applications and services within the containment framework. The drawing is exemplary, and various layers may be added, combined, or omitted as appropriate for a particular device or implementation.

[0038] The base layer shown in Figure 3 is the device hardware layer 120, which comprises the hardware resources necessary to support a software system, such as a processor and system memory. In one embodiment, the hardware of a small footprint device, such as the small footprint device hardware example illustrated in Figure 2, implements the hardware layer 120 illustrated in Figure 3. However, in other embodiments, the hardware layer 120 may be implemented in other types of devices, such a device with even greater resource constraints than a typical small footprint device, such as a smart card.

[0039] As shown in Figure 3, the next layer up from the hardware layer is the operating system layer 122. As is well known in the art, the operating system functions as an interface layer between the device hardware and software running on the device and serves as a manager for low-level tasks such as input/output, memory management, etc. The operating system 122 illustrated in Figure 3 may be any particular operating system which supports the higher layers shown in Figure 3. The operating system 122 may be a small and efficient one that is suitable for or written particularly for use in a small footprint device. For example, the operating system 122 may be the Java™ OS operating system available from Sun Microsystems, Inc.

[0040] In one embodiment, the containment framework is implemented in a Java™ application environment as one or more Java™ classes. As shown in Figure 3, the Java™ virtual machine layer 124 and Java™ application programming interface (API) class libraries layer 126 are the next layers up from the operating system. These two layers together make up the Java™ application environment, or Java™ platform. Classes implementing the containment framework may be built using the Java™ libraries 126 and compiled into bytecodes. The bytecodes are instructions which execute on the Java™ virtual machine 124, which interacts with the operating system 122 and/or the device hardware 120.

[0041] In one embodiment, the containment framework is implemented in the PersonalJava™ Java™ application environment, which is a Java™ platform designed to be highly scalable, modular, and configurable, while requiring minimal system resources. PersonalJava™ comprises the Java™ virtual machine and a subset of the Java™ API, including core and optional APIs and class libraries. In addition, the PersonalJava™ API includes specific features required by consumer applications in resource-limited environments, such as a specialized version of the Java™ abstract window toolkit (AWT). The PersonalJava™ AWT library is targeted and tuned for consumer product look and feel, providing graphics and windowing features while supporting low-resolution displays and alternate input devices (via an extended event model for mouse- and keyboard-less devices).

[0042] Referring again to Figure 3, the containment framework 128 is shown as the next layer up from the Java™ platform layer. As noted above, the containment framework 128 may also be based on other platforms. As described in detail below, the containment framework 128 manages program modules, e.g. by enabling module registration, lookup, instance tracking, etc. Modules may provide various services. The containment framework 128 enables modules to request other modules, in order to use their services. Applications may be implemented as modules that utilize the services of other modules. The containment framework 128 thus provides a lightweight, extendable service and application framework, enabling applications to coexist and share a modular code base.

[0043] This type of extendable architecture enabling multiple program modules to cooperate is an important development for small footprint devices. Small footprint devices have historically been limited to relatively narrow uses. For example, cellular phones were typically used for telephony and little else. However, as various technologies are developed allowing small footprint devices to become "smarter", having general-purpose processors, larger display screens, etc., it has become desirable to expand the scope of applications used in small footprint devices.

[0044] The present containment framework may enable the types of applications and services generally associated with desktop computing environments to work together in a small footprint device, in a manner that desktop computer users are familiar with. As illustrated in Figure 3 and described above, services and applications 130 running on a small footprint device may be implemented as modules built on the containment framework layer 128. For example, the Personal Applications suite available from Sun Microsystems, Inc. is built using one embodiment of the containment framework 128. The Personal Applications Suite comprises an integrated set of applications such as a browser, an email client, and a personal organizer.

[0045] Figure 3 also illustrates the ability of some embodiments of the containment framework 128 to integrate off-device services 132 with on-device applications/services 130. For example, the containment framework 128 may provide an interface between a small footprint device and a network such as a Jini™ network. A small footprint device system may register its services for use by other devices or clients in a network. The containment framework may also enable services and applications within the small footprint device to look up and use services provided by other network devices. The integration of services of the small footprint device with network services is discussed in more detail below for Figure 4.

Figures 4 - 8: Exemplary Network Device and Service Federation

[0046] Figure 4 illustrates an exemplary network in which a small footprint device running applications/services in the containment framework is connected to a local service-based network. In the example shown, a smart cellular phone 134 utilizing the containment framework 144 is connected to the network. Also shown attached to the network are a printer 130 and an internet-enabled television 132. In this example, it is assumed that the printer 130 and television 132 devices are operable to export services to a network and possibly use the services of other devices on the network. For example, the printer may export its print service 138, and the internet television may look up the print service and use it to print a web page. To facilitate the federation of devices and services in this manner, a lookup service 136 is located on the network. The lookup service 136 may reside on a separate device such as a network server.

[0047] The federation of devices and services may be implemented in various ways. For example. Jini™ technology, available from Sun Microsystems, Inc., comprises components and a programming model which enables the type of distributed system illustrated in Figure 4. In one embodiment, the local network shown in Figure 4 may be a Jini™ network, and the printer 130 and internet television 132 may be Jini™ - enabled devices. Each device is operable to find the Jini™ network lookup service and register the services it offers with the lookup service. The lookup service maps interfaces indicating the functionality provided by a service to sets of objects that implement the service.

[0048] To add its services to a service federation, a device or other service provider may first locate an appropriate lookup service by using a "discovery" protocol. Figure 5 illustrates the discovery process. As shown, the service provider 164, e.g. the printer 130 shown in Figure 4, may broadcast a request on the local network for any lookup services to identify themselves.

[0049] Once the service provider 164 has located the lookup service 160, the service provider 164 may register its service with the lookup service 160 by using a "join" protocol. Figure 6 illustrates the join process. The service provider 164 may create a service object which clients can use to invoke the service. As illustrated in Figure 6, the service object for the provided services may then be loaded into the lookup service 160, along with service attributes or descriptors containing information about the types or names of services provided. For example, in a Jini™ network, the printer 130 shown in Figure 4 may create a service object which comprises a Java™ programming interface for the print service 138. The printer 130 may then call a "register" method of the lookup service 136, passing this service object, along with attributes which specify that the service 138 being registered is a print service, the printing resolution, the possible paper sizes, etc.

[0050] Once the service provider 164 has joined its services with the lookup service 160, other network clients may request and use the services. The process of requesting a service, called lookup, is illustrated in Figure 7. After discovering the lookup service, a client 162 may request a service from the lookup service 160 using a description of the requested service. The lookup service 160 attempts to match the description given by the requestor to the services that have joined the lookup service. The lookup service 160 may use the service attributes sent by the service provider 164 during the join process to perform this matching. If a match is found, the lookup service 160 provides the appropriate service object to the client 162. For example, a Java™ interface for the requested service may be provided to the client 162.

[0051] Once a client 162 has received a service object from the lookup service, the client may invoke the service. Figure 8 illustrates the process of service invocation. When a service is invoked, the client 162 and the service provider 164 may communicate directly with each other. Any of various interaction protocols may be used for this communication. For example, the protocol used may be Java™ Remote Method Invocation (RMI), CORBA, DCOM, etc. The service object that a client receives from the lookup service may call back to code located at the service provider, e.g. by calling an RMI method, or it may execute locally to provide the requested service, or it may use a combination of these approaches.

[0052] As shown in Figure 4, the lookup service 136 for a local network may also act as a gateway to an outside network such as the Internet 154. The service-based distributed computing model may thus be extended to include clients and services located outside the local network. For example, the technology being developed for the Open Service Gateway Initiative (OSGI) may be leveraged to implement this type of distributed computing system.

[0053] This type of service sharing between and across different networks and the Internet may enable new types of applications to be developed. For example, merchants may use Internet services to record data about specific consumers, and advertising service providers may use this data to push context-specific ads onto consumer devices, depending on which local network the device is connected to, etc. For example, a customer may enter a shopping mall and connect a personal data assistant (PDA) into a local network for the shopping mall, via a wireless connection. An Internet-based consumer data service may be joined with the lookup service for the shopping mall network and may provide information about the specific consumer who has just plugged into the mall network. Services running in the shopping mall network may then use this data together with other factors such as the customer's current location within the mall, the time of day. etc., in order to generate personalized ads and push them onto the customer's PDA.

[0054] Many other examples of services based on the network of Figure 4 are possible. For example: network-enabled consumer devices within a home may utilize a service provided by a power company, via the Internet, which manages power consumption within the home; security service providers may monitor a home or specific devices via network services and may notify the owner immediately when property is broken into; health service providers may remotely monitor a patient's state by communicating with medical instruments; etc.

[0055] In the examples listed above, an assumption is made that devices are able to transparently connect to a network, integrate network services with device-resident services, and export device-resident services for use by network clients. The containment framework described herein may provide the necessary interface to integrate services and applications of small footprint devices such as personal data assistants, handheld computers, smart cellular phones, etc. with a network service federation.

[0056] As shown in Figure 4 and described in more detail below, the containment framework 144 has its own type of lookup service 146. The lookup service 146 within the containment framework 144 may operate similarly to the local network lookup service described above, utilizing discovery, join, lookup, and service invocation processes. For example, the personal organizer application 152 may utilize various services such as a calendar service, a contact list service, a bookmark service, etc. (not shown). The personal organizer application 152 may obtain a reference for communicating with these services via the containment framework lookup service 146.

[0057] The containment framework 144 may integrate its own lookup service 146 with an off-device lookup service such as the local network lookup service 136 shown in Figure 4. In this way, off-device services such as the print service 138 and the web service 140 may become available to the applications/services 148, 150, and 152 of the containment framework, and vice versa. For example, the personal organizer application 152 may request a print service from the containment framework lookup service 146. The containment framework lookup service 146 may first search for an on-device print service. If one is not found, the containment framework lookup service 146 may then request a print service from the network lookup service 136. The service object for the print service 138 may then be returned to the personal organizer 152. An interface 142 between the on-device services/applications and the off-device services is illustrated in Figure 4. Details follow on how the integration of on-device/off-device services may be implemented.

[0058] As noted above, clients of services may themselves be services to other clients. For example, the email client "application" 150 of the smart cellular phone shown in Figure 4 may itself be a service to a client running in the containment framework 144 or to a network client. For example, in the case of malfunction, the printer 130 shown in Figure 4 may request an email service so that it can send diagnostic information to a service technician. If the network lookup service 136 cannot find a network-based email service, it may request an email service from the smart cellular phone 134 via the interface 142. A service object for the email application/service 150 running in the containment framework 144 may be passed to the requesting printer client 130. In this example, the printer client 130 may communicate directly with the email application/service 150 to send an email containing diagnostic information to a printer service technician. The email application/service 150 may send the email immediately if it is able to find an email server service, or it may send the email later when such a service becomes available when the cellular phone user connects to a different network.

[0059] Although the above description references specific protocols and programming models, such as Jini™ technology, it is noted that these specific technologies are exemplary only. For example, the applications and services within the containment framework may be integrated with clients, services, devices, networks, etc. which employ any of various types of standards, protocols, and programming models, including, but not limited to: Jini™, CORBA, COM/DCOM, Bluetooth, CAL, CEBus, HAVi, Home API, HomePNA, HomePnP, HomeRF, VESA, etc.

Figure 9 - Containment Framework Block Diagram

[0060] Figure 9 is an abstract block diagram illustrating the basic architecture of the containment framework environment. As described above, the containment framework provides a containment system for applications and services. These applications and services are managed within the system as units called modules. The containment framework is lightweight; in one embodiment, modules may interact with a single framework manager object which performs all module management. This manager is referred to herein as the central framework instance. In one embodiment, the central framework instance may be implemented as an instance of a Java™ class. Figure 9 illustrates the central framework instance 170 and the code and data it comprises/manages. It is noted that Figure 9 illustrates one embodiment of the containment framework. Other embodiments may employ a different architecture and/or may be implemented in different programming languages or software environments. For example, the module management/containment performed by the central framework instance 170 illustrated in Figure 9 may be performed by multiple objects or components in other embodiments.

[0061] As shown in Figure 9, the central framework instance 170 comprises data 182 representing the modules currently loaded in the system. The containment framework architecture is non-hierarchical. Thus, the loaded modules may be represented as a flat list or array of modules. This non-hierarchical system helps to keep the core containment framework code and the modules running within the framework compact. Systems employing hierarchical components such as Java-

Beans™ components may provide associated benefits, but the benefits come at the expense of a more complex management system requiring more system resources. However, the containment framework does provide a mechanism for the non-hierarchical modules to gain many of the benefits of a hierarchical containment system. This mechanism is described below for Figures 10 and 11.

[0062] As shown in Figure 9, in one embodiment the central framework instance 170 comprises publicly accessible methods 178 which modules may call. These methods may be broken into abstract groups. For example, one group of methods 172 may comprise lookup methods. Lookup methods implement the lookup service functionality described above. Modules may pass a module descriptor to a lookup method of the central framework instance 170 to locate a particular service module. The containment framework lookup process is described below for Figure 13. Another group of framework methods 174 may comprise methods for loading and unloading modules. After finding a service module, a client module may request the central framework instance 170 to load the service module and return a reference to the loaded module. The client module may then invoke the service. The client may call a framework method to release the service module when it is finished using it. Although described as distinct groups, the division of methods into lockup and load/unload groups may be only a conceptual division. For example, in one embodiment a lockup method may also load a module that it matches and return a reference to the matched module.

[0063] Figure 9 also illustrates system data 180 referred to as framework metadata, which may comprise data 182 describing the list of loaded modules and other data describing the state of the system. Another abstract group of methods 176 of the central framework instance 170 may comprise reflection methods. Reflection methods are somewhat different than the other groups of methods since they provide direct access to the core metadata 180. A special class of modules called system modules may call reflection methods to gain access to the metadata 180. Regular modules may not access the metadata 180.

[0064] After receiving a reference to the core system data 180, a system module may use or modify the data in any way desirable. Thus, the containment framework is highly extendable. The central framework instance 170 may itself remain small, and system modules may be added to implement any functionality not already enabled by the central framework instance 170. For example, a system module may enable the integration described above for Figures 4 - 8 between applications/services running within the containment framework and services based in an external network.

[0065] In this example, such a system module may be written as a secondary lookup service that conforms to the protocols and programming model of the external network. For example, for a Jini™ network, a system module may be written which discovers the Jini™ network lookup service and joins the network lookup service, registering itself as a secondary lookup service. When a network client requests a service, the network lookup service may invoke the lookup service implemented by the system module. This system module may attempt to find a service module within the containment framework which matches the description of the requested service. If a match is found, then the system module may perform any necessary steps to export the service module to the network client, since the system module has full access to the system module list and metadata. For example, the system module may load and register the matched service module into the system and return an interface, such as a Java™ interface, to the newly loaded module to the requestor.

Figures 10 and 11 - Simulating a Hierarchical Environment

[0066] It is often desirable to establish a hierarchical context for modules. For example, several service modules of the same type may be present in a system, but each may behave slightly differently. In a hierarchical containment system, a request by a module for a service may be filtered through a parent or containing module of the requesting module so that a reference to a specific service module may be passed back to the requestor. Hierarchical containment also has other inherent advantages, such as an ability to easily distribute and store data among a hierarchy of modules. However, as stated above, a full implementation of a hierarchical containment system may be very costly in terms of the system resources required, such as memory and processing power. The containment framework may provide a mechanism giving developers and applications many of the benefits of hierarchical containment, but without the high overhead costs usually associated with it.

[0067] For example, one embodiment of the containment framework allows modules to register themselves as module request listeners of other modules. For example, a module A may register itself as a request listener of a module B, e.g., by calling an AddRequestListener method of the central framework instance. When module B subsequently calls a method of the central framework instance to find a particular service, the central framework instance checks for any module request listeners for module B. In this case, it finds module A as a request listener, and asks module A to provide the requested service module to module B.

[0068] Figures 10 and 11 illustrate an exemplary use of module request listeners in the containment framework. Figure 10 illustrates a desired conceptual module hierarchy for print services. As shown in the figure, two print service modules 192 and 194, print service A and print service B, are encapsulated in a print manager module 190. For example, the two print services 192 and 194 may print to different locations, have different resolution and color capabilities, etc. Either of these print service modules may satisfy a lookup request made by another module for a print service. However, it may be desirable to employ a print manager module which selects and returns a particular print service. For example the print manager 190 may select a print service based on which client module makes the print request, or the print manager may display a dialog box asking for user input for the desired print service characteristics.

[0069] Although the containment framework utilizes a non-hierarchical containment model, the hierarchy illustrated in Figure 10 may be realized by registering the print manager module 190 as a module request listener of client modules that may request a print service. Figure 11 illustrates example modules 198 which may run in a system. As described earlier, these modules may themselves employ other modules as services. According to the non-hierarchical model of the containment framework, the modules are shown arranged in a flat layout, with no inherent module hierarchy.

[0070] In this example, the web browser module 196 may be operable to make a print request, e.g., for printing a web page. As shown in Figure 11, the print manager module 190 may be registered as a module request listener for the web browser module 196. Upon receiving the print service request from the web browser 196, the containment framework lookup service may find the print manager module 190 registered as a request listener for the web browser module 196 and may ask the print manager module 190 to provide a print service module to the web browser requestor 196. The print manager module 190 may then return a reference to print service module A 192 or print service module B 194, or the print manager module 190 may present a dialog box to the user to decide which print service module to return, etc. Thus, the desired module hierarchy of Figure 10 may be implemented for non-hierarchical modules of the containment framework.

Figure 12 - Parcel Packaging Units

[0071] Modules may be packaged into units referred to as parcels. This packaging serves several purposes. For example, parcels provide a convenient mechanism to manage related code and data as a unit. If closely related modules have static dependencies, then they may be packaged together into a parcel. Parcels may be used to handle installation and upgrading within a system.

[0072] Figure 12 illustrates an example parcel 200 that groups together modules related to a personal information manager (PIM). The figure shows a calendar module 202, a contact list module 204, an appointment module 208, and a user interface module 206. Various other modules may be present in the parcel as desired. The modules of the PIM parcel 200 may also make use of various core service modules running within the containment framework, such as bookmark services, find services, etc. The use of a PIM parcel may simplify installation and upgrading of a PIM application. Packaging the PIM modules into a parcel in this way also has the development-time benefit of creating separate code units for multi-target development.

[0073] Parcels also provide an additional way to provide a run-time context for non-hierarchical modules. When a module is loaded into the system, the central framework instance may store metadata specifying which parcel, if any, the module belongs to. Service modules may later use this information to provide services differently for different client modules, depending on which parcel the client belongs to. For example, client modules may use a file access service module to obtain a root directory. The file access module may return different root directories for different clients, depending on which parcels the clients belong to.

Figure 13 - Module Request Flowchart Diagram

[0074] Figure 13 is a flowchart diagram illustrating a typical lookup process that the central framework instance may perform when it receives a lookup request for a service module from a client module. It is noted that Figure 13 is exemplary and that various steps may be combined, omitted, or modified. For example, as noted previously, system modules may be added which customize the lookup process.

[0075] In step 300 of Figure 13, the central framework instance receives a module lookup request from a requestor module. For example, the requestor module may call a RequestModule method of the central framework instance, passing a module descriptor for the service module being requested, as well as a reference to the requestor module itself. The reference to the requestor module may be added to the system data so to keep track of service module users. As described in more detail below, a module may be unloaded when no other modules are using it.

[0076] The module descriptor passed by the requestor module specifies various attributes about the requested module that the framework instance can use to attempt to find a matching module. This module descriptor may be an object which comprises information such as the requested module's service type, class name, and/or service-specific attributes, etc. The requestor may also pass a text description to the central framework instance, which the central framework instance may use to create a module descriptor object.

[0077] In step 302, the central framework instance checks to see whether any request listener modules are registered for the requesting module. If a request listener is found, then in step 304 the framework instance notifies the request listener of the request and instructs the request listener to attempt to provide a module which matches the module request descriptor. If the request listener can provide a matching module, then execution proceeds to step 314. Otherwise, other registered request listeners may be asked to provide a module, until a match is found or there are no more request listeners.

[0078] If no request listeners are found, or if no request listeners can provide the requested module, execution proceeds to step 306. However, in one embodiment, if one or more request listeners are registered for the requesting module, and none of them are able to provide a matching module, then execution may stop after step 304. In step 306, the central framework instance checks the list of modules to determine whether one of the modules matches the module descriptor. If a match is found, then in step 308 the framework instance checks whether the matched module is multi-instantiable. If not, then execution proceeds to step 314.

[0079] If the matched module is found to be multi-instantiable in step 308, then the central framework instance may continue to search through the module list for a match. If there are no more modules to search, execution proceeds to step 310. In step 310, the framework instance searches for module-provider modules in the module list. Module-provider modules are modules capable of providing a requested module. For example, a network lookup service may be imported as a module-provider module for the containment framework.

[0080] If a module-provider module is found, then in step 312, the central framework instance notifies the module-provider module of the request and instructs it to attempt to provide a module which matches the module request descriptor. If a match is found then execution proceeds to step 314. If the module provider cannot provide the requested module, the central framework instance may search for other module-provider modules and repeat step 312. If no module providers are present in the module list or if none can provide the requested module, then the requestor is notified that the request cannot be fulfilled, and execution completes.

[0081] Step 314 may be reached from step 304, 308, or 312. In all cases, a module is found which matches the module request descriptor. In step 314 the requestor is registered as a user of the matched module, and in step 316 a reference to the matched module is returned to the requestor. Any necessary initialization steps involved in loading and initializing the matched module are also performed in step 314. For example, modules may have an Initialize method that is called when a module is loaded.

[0082] As noted above, the flowchart of Figure 13 is exemplary, and various embodiments may have different lookup/load scenarios. For example, a module may call a central framework method to load a service module without returning a reference to the matched module, or request listeners may be ignored in some cases, etc.

Figure 14 - Module Release Flowchart Diagram

[0083] When a client module is finished using a service module, the client may call a method of the central framework instance to release the module. Figure 14 is a flowchart diagram illustrating the module release process. The flowchart of Figure 14 is exemplary, and various steps may be combined, omitted, added, or modified as required or desired for different embodiments.

[0084] In step 330, the central framework instance receives a module-release notice from a user module. As described above for Figure 13, when a user module requests a service module, the user module is added to a list of users of the service module. In step 332, the central framework instance removes the releasing user module from the list of users of the released module. In step 334, the framework instance determines whether any other user modules are using the released module, e.g., by checking whether other modules are present in the releases module's user module list. If so, then execution stops.

[0085] If no other modules are using the released module, the central framework instance may attempt to unload the released module. In step 336, the framework instance may call a CanFinalize method of the released module. The CanFinalize method returns true if the module can be unloaded, or false otherwise. If the CanFinalize method returns false in step 336, then execution stops. Otherwise, a Finalize method of the released module may be called. The Finalize method may perform any necessary steps for unloading the module, such as releasing resources. The module may then be unloaded, which may involve garbage-collection, etc., depending on the particular embodiment.

[0086] A computer program product for implementing the invention can be in the form of a computer program on a carrier medium. The carrier medium could be a storage medium, such as solid state magnetic optical, magneto-optical or other storage medium. The carrier medium could be a transmission medium such as broadcast, telephonic, computer network, wired, wireless, electrical, electromagnetic, optical or indeed any other transmission medium.

[0087] Although the present invention has been described in connection with specific embodiments, it is not intended to be limited to the specific forms set forth herein, but on the contrary, it is intended to cover such alternatives, modifications, and equivalents, as can be reasonably included within the scope of the invention as defined by the appended claims.

Claims

1. A method of caching a web page in a device, the method comprising:

receiving a web page in response to a request;

creating a reduced form of the web page;
caching the reduced form of the web page.

2. The method of claim 1, wherein said request is a request to view said web page; the method further comprising:

displaying the web page after receiving the web page;
using the reduced form of the web page to redisplay the web page in response to a subsequent request to view the web page.

3. The method of claim 1, wherein said creating a reduced form of the web page comprises removing particular markup language tags from the web page.

4. The method of claim 3, wherein said markup language tags control properties affecting display of the web page.

5. The method of claim 4, wherein said properties affecting display of the web page comprise properties selected from the group consisting of:

color, font, and font size.

6. The method of any of claims 3 to 5, wherein said markup language is HTML.

7. The method of any of claims 3 to 6, wherein said markup language tags control display properties which are not supported by said device.

8. The method of any preceding claim, wherein said creating a reduced form of the web page comprises removing scripting code from the web page.

9. The method of claim 8, wherein said scripting code is JavaScript™ code which is intended to dynamically change the appearance of a web page.

10. The method of any preceding claim, wherein said creating a reduced form of the web page comprises parsing the web page into a tree data structure, and wherein said caching the reduced form of the web page comprises caching the tree data structure.

11. The method of claim 10, wherein said tree data structure is a tree comprising elements representing elements of the web page, wherein said elements of the tree are related to each other according to the Document Object Model.

12. The method of claim 2, wherein:

said displaying the web page after receiving the web page comprises parsing the web page into a tree data structure and accessing the tree data structure to determine how the web page is displayed;
said caching the reduced form of the web page comprises caching said tree data structure; and
said using the reduced form of the web page to redisplay the web page comprises accessing said tree data structure from the cache to determine how the web page is displayed.

13. The method of any preceding claim, wherein said device is a small footprint device.

14. The method of any preceding claim, wherein said device is a device from the group consisting of:

personal data assistant (PDA), cellular phone, and global positioning system (GPS) receiver.

15. A device operable to cache a reduced form of a web page; the device comprising:

a processing unit;
system memory coupled to said processing unit;
a software program stored in said system memory, wherein said software program is operable to:
receive a web page in response to a request;
create a reduced form of said web page;
cache said reduced form of said web page.

16. The device of claim 15, wherein said creating a reduced form of the web page comprises removing particular markup language tags from the web page, wherein said markup language tags control properties affecting display of the web page.

17. The device of claim 16, wherein said properties affecting display of the web page comprise properties selected from the group consisting of:

color, font, and font size.

18. The device of claim 16 or claim 17, wherein said markup language is HTML.

19. The device of any of claims 15 to 18, wherein said markup language tags control display properties which are not supported by said device.

20. The device of any of claims 15 to 19, wherein said software program is further operable to:

display the web page after receiving the web page, wherein said displaying the web page

after receiving the web page comprises parsing the web page into a tree data structure and accessing the tree data structure to determine how the web page is displayed;
use the reduced form of the web page to redisplay the web page in response to a subsequent request to view the web page wherein said using the reduced form of the web page to redisplay the web page comprises accessing said tree data structure from a cache to determine how the web page is displayed.

21. The device of any of claims 15 to 20, wherein said device is a small footprint device.

22. A computer program comprising program instructions for caching a web page, wherein the program instructions are executable to implement:

receiving a web page in response to a request;
creating a reduced form of the web page;
caching the reduced form of the web page.

23. The computer program of claim 22, wherein said request is a request to view said web page, and wherein the program instructions are further executable to implement:

displaying the web page after receiving the web page;
using the reduced form of the web page to redisplay the web page in response to a subsequent request to view the web page.

24. The computer program of claim 22 or claim 23, wherein said creating a reduced form of the web page comprises removing particular markup language tags from the web page, wherein said markup language tags control properties affecting display of the web page.

25. The computer program of any of claims 22 to 24, wherein said properties affecting display of the web page comprise properties selected from the group consisting of:

color, font, and font size.

26. The computer program of any of claims 22 to 25, wherein said creating a reduced form of the web page comprises parsing the web page into a tree data structure, and wherein said caching the reduced form of the web page comprises caching the tree data structure.

27. The computer program of any of claims 22 to 26, wherein said tree data structure is a tree comprising elements representing elements of the web page, wherein said elements of the tree are related to each other according to the Document Object Model.

28. The computer program of any of claims 22 to 27, wherein:

said displaying the web page after receiving the web page comprises parsing the web page into a tree data structure and accessing the tree data structure to determine how the web page is displayed;
said caching the reduced form of the web page comprises caching said tree data structure; and
said using the reduced form of the web page to redisplay the web page comprises accessing said tree data structure from the cache to determine how the web page is displayed.

29. The computer program of any of claims 22 to 27 on a carrier medium.

30. The computer program of claim 29, wherein the carrier medium is a storage medium.

31. The computer program of claim 29, wherein the carrier medium is a transmission medium.

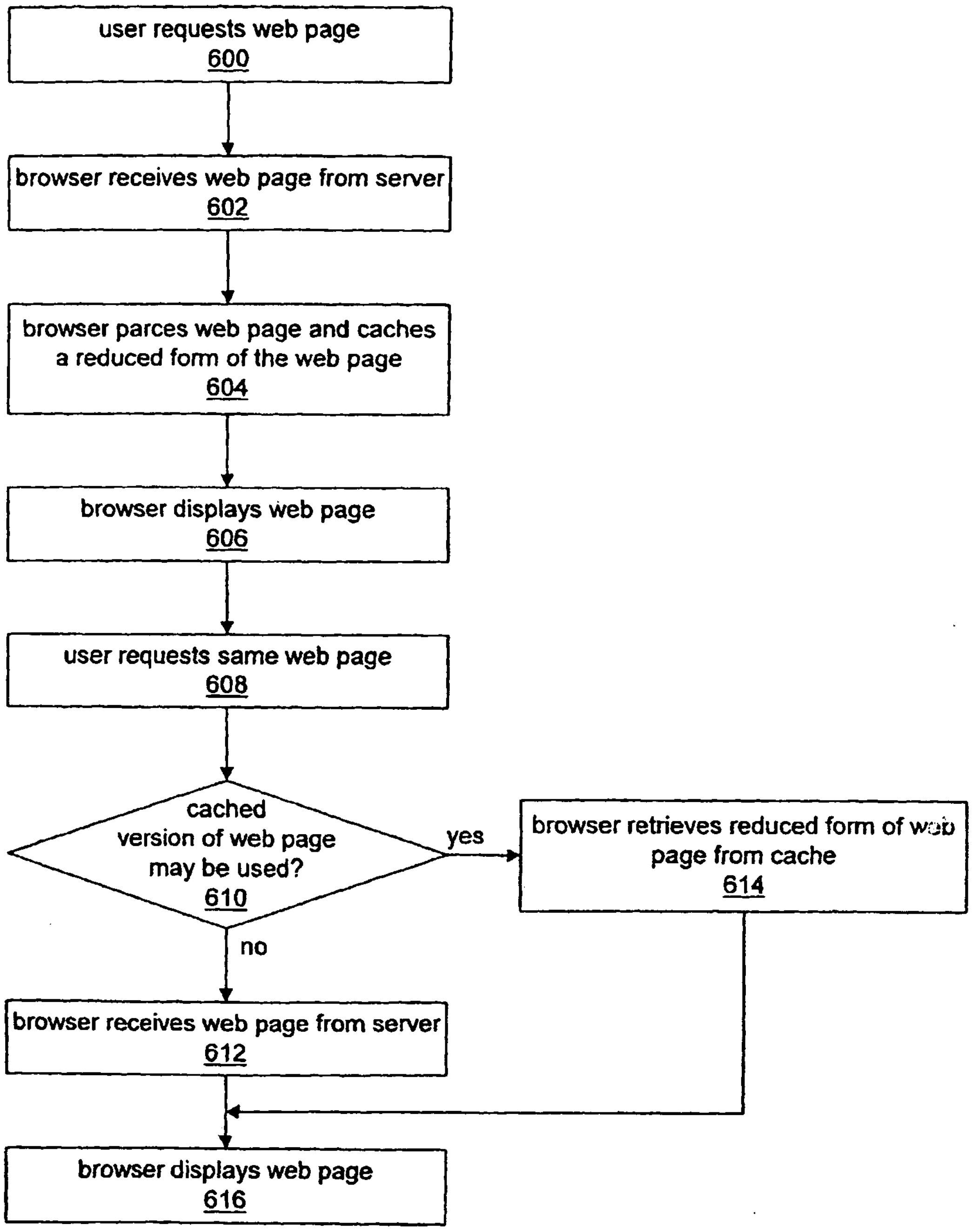
user requests web page
600
browser receives web page from server
602
browser parces web page and caches a reduced form of the web page
604
browser displays web page
606
user requests same web page
608
cached version of web page may be used?
610
yes
browser retrieves reduced form of web page from cache
614
no
browser receives web page from server
612
browser displays web page
616

Fig. 1

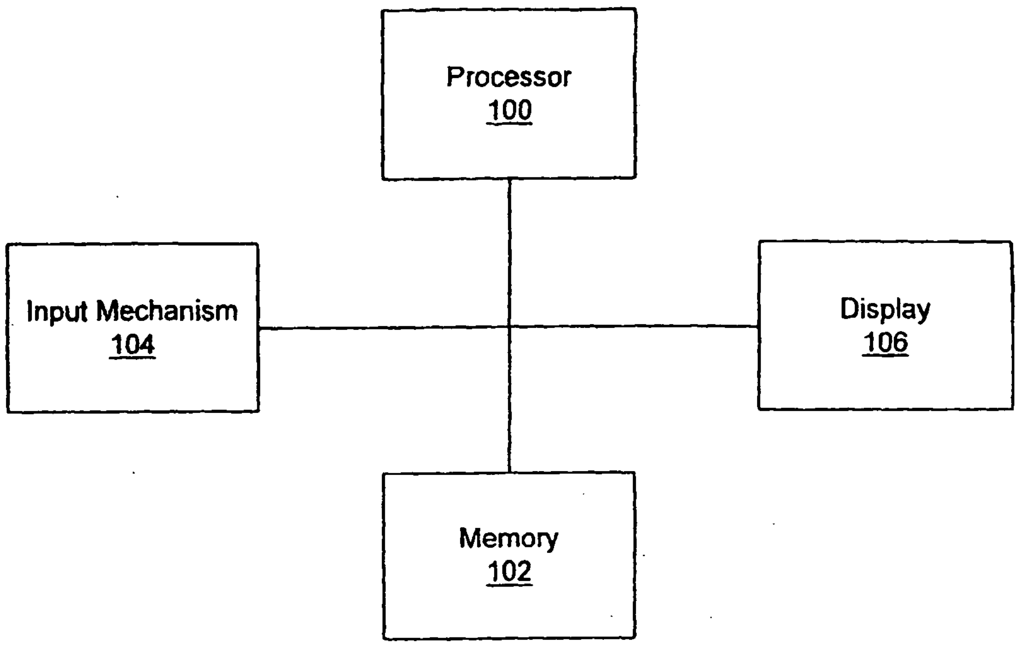
Processor
100
Input Mechanism
104
Display
106
Memory
102

Fig. 2

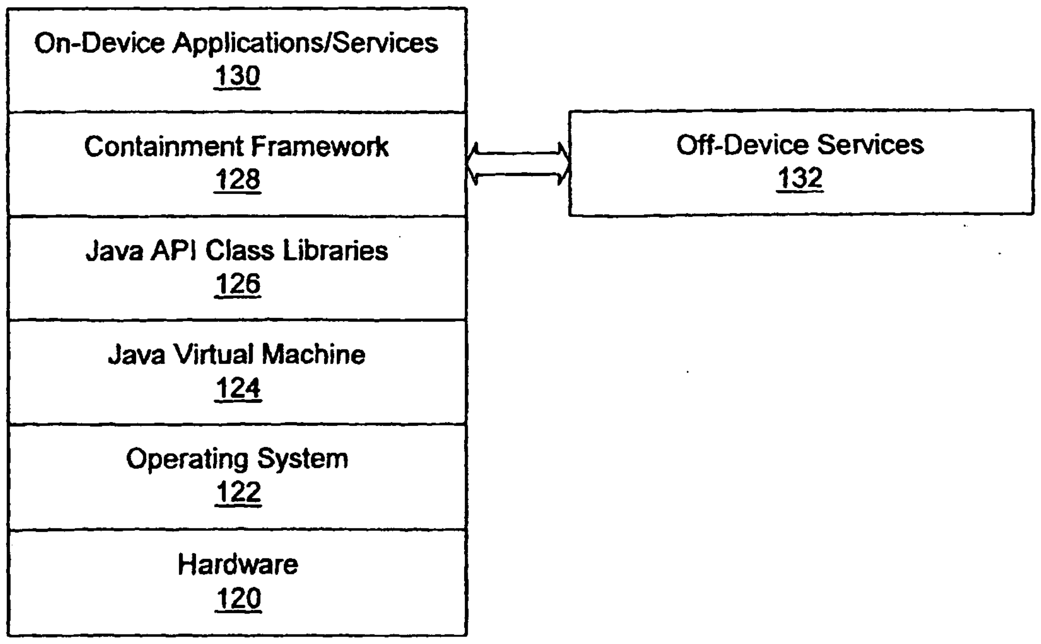
On-Device Applications/Services
130
Containment Framework
128
Java API Class Libraries
126
Java Virtual Machine
124
Operating System
122
Hardware
120
Off-Device Services
132

Fig. 3

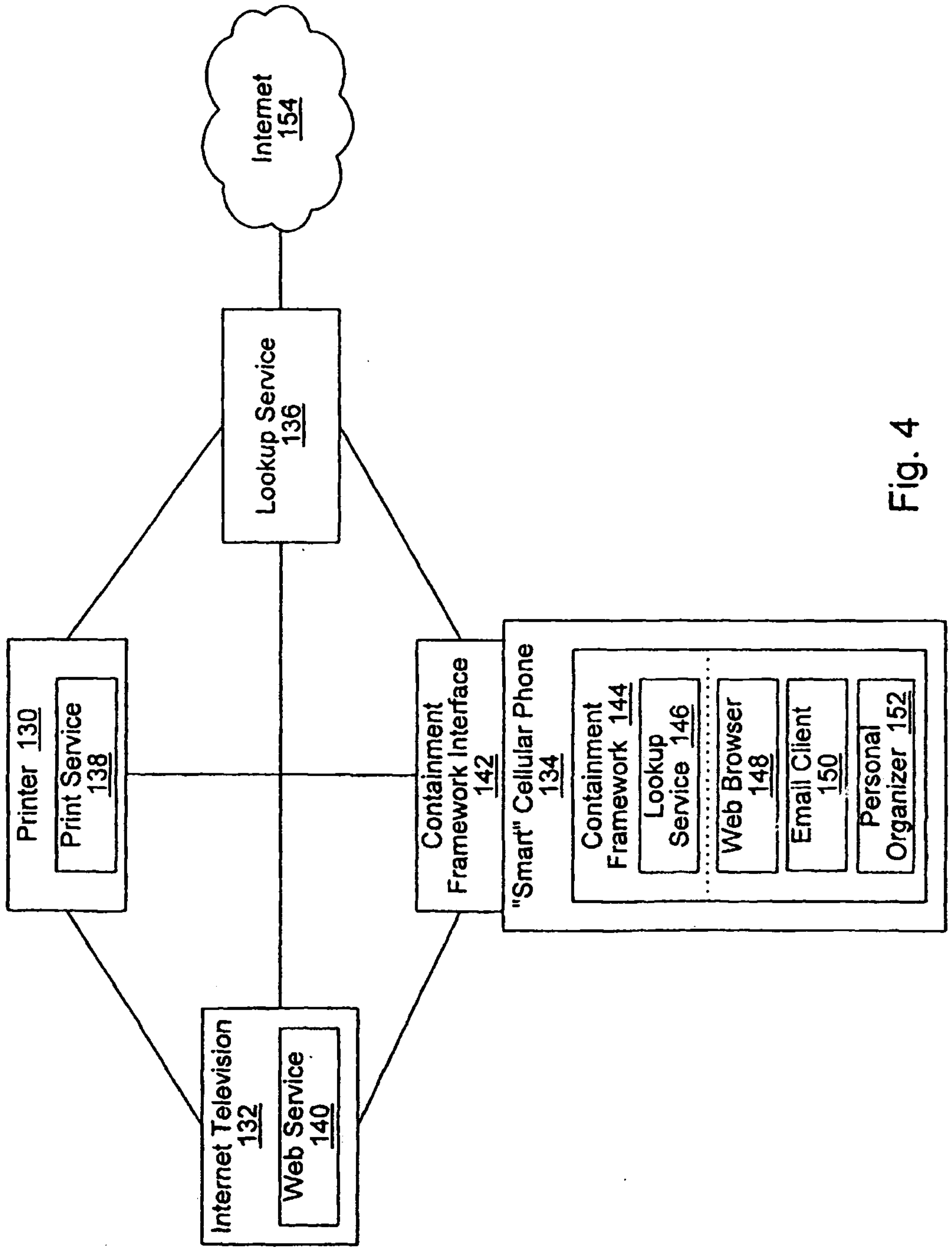
Internet 154
Lookup Service 136
Printer 130
Print Service 138
Containment Framework Interface 142
"Smart" Cellular Phone 134
Containment Framework 144
Lookup Service 146
Web Browser 148
Email Client 150
Personal Organizer 152
Internet Television 132
Web Service 140

Fig. 4

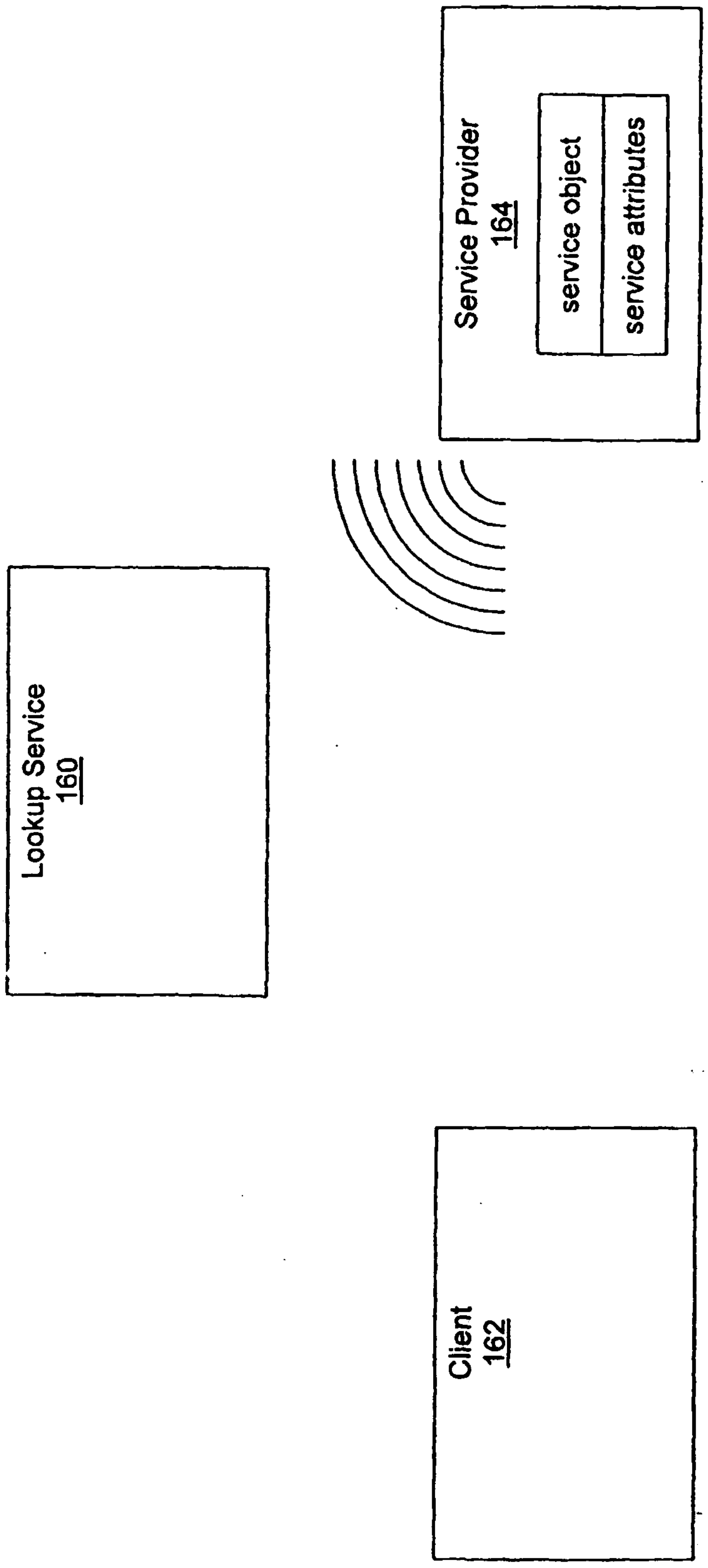
Service Provider
164
service object
service attributes
Lookup Service
160
Client
162

Fig. 5

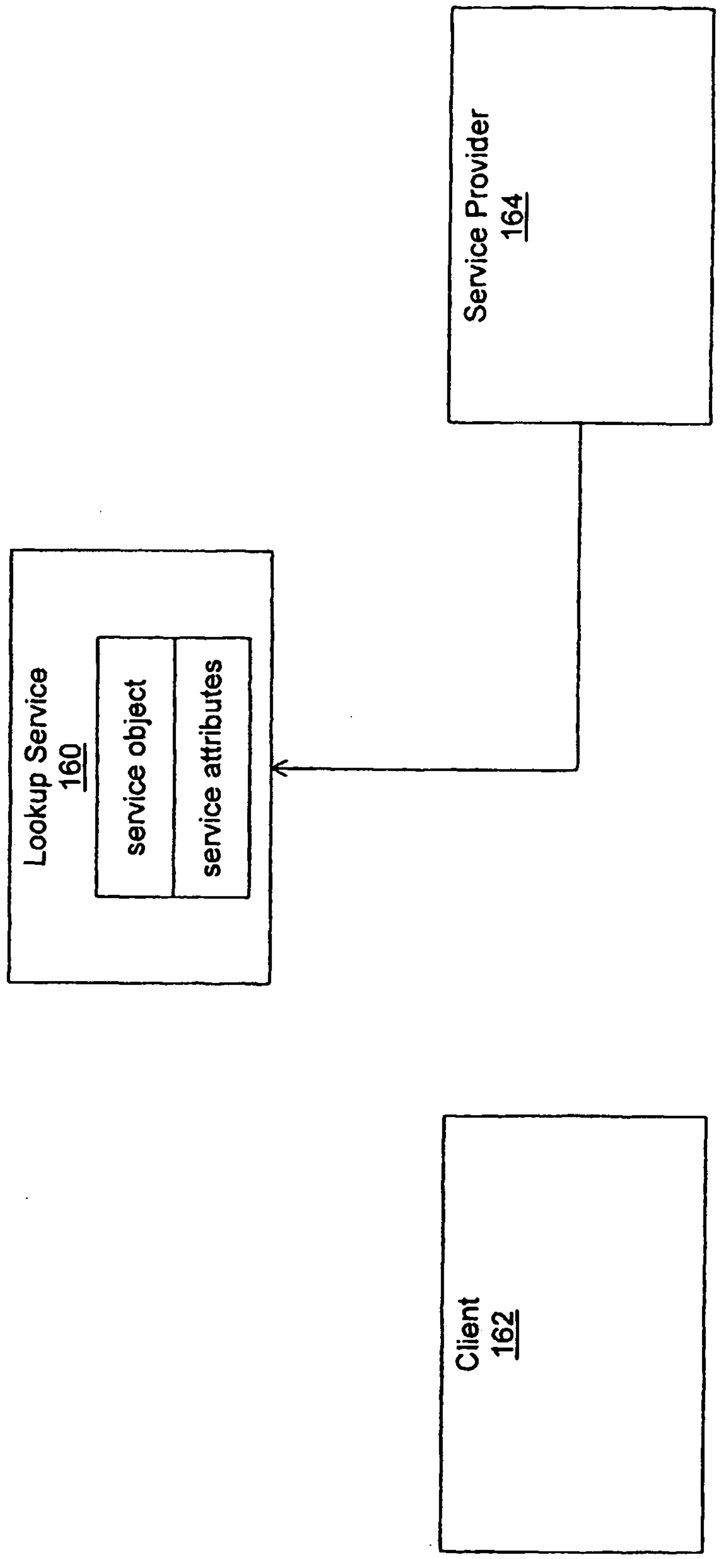
Service Provider
164
Lookup Service
160
service object
service attributes
Client
162

Fig. 6

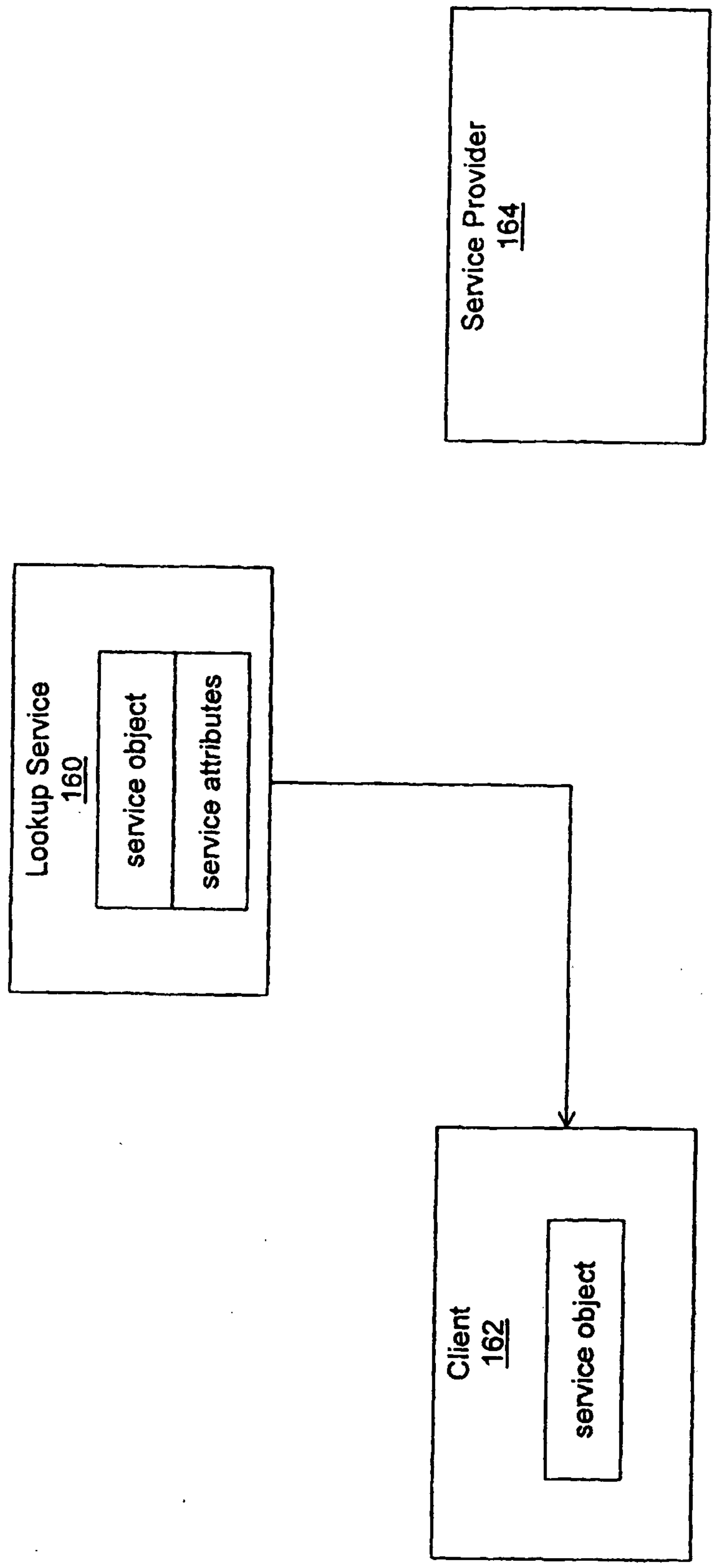
Service Provider
164
Lookup Service
160
service object
service attributes
Client
162
service object

Fig. 7

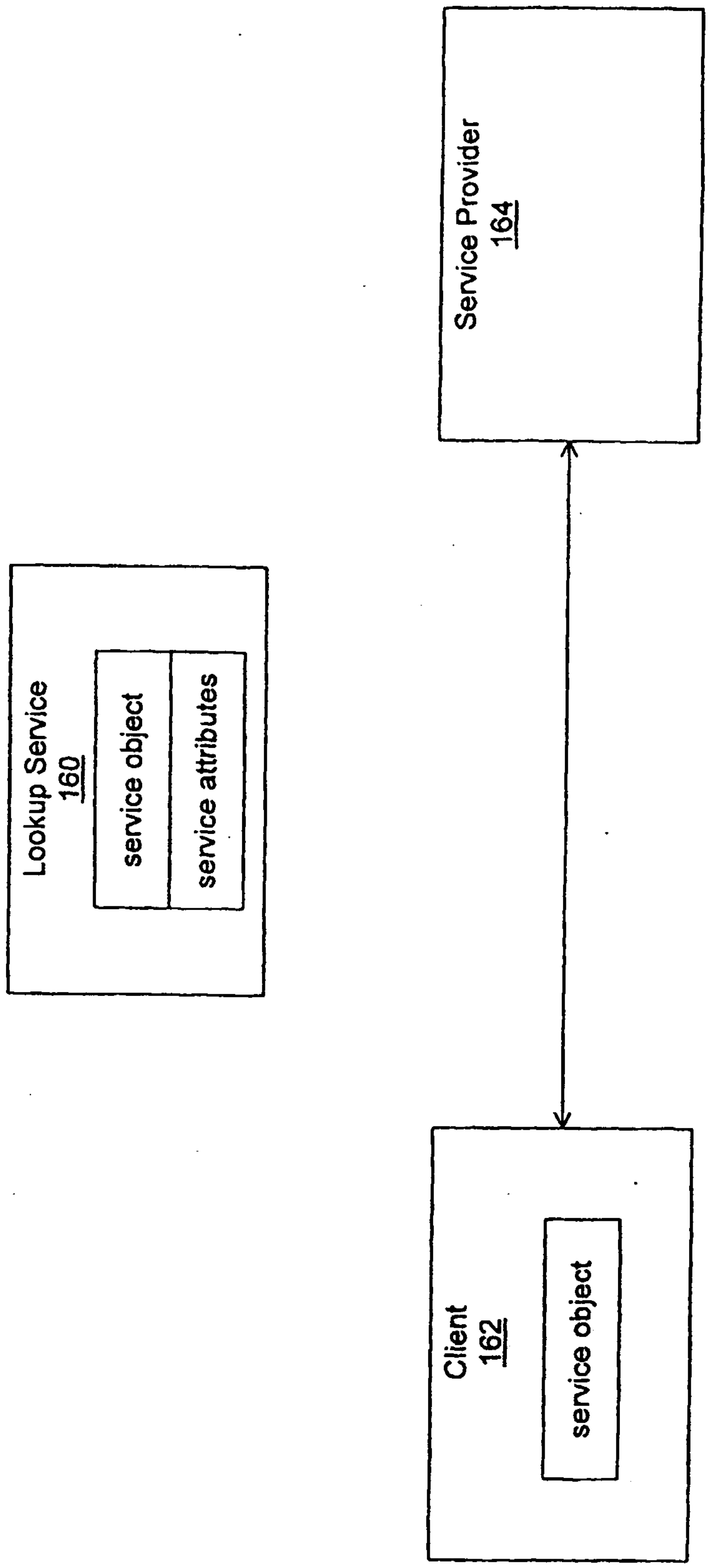
Lookup Service
160
service object
service attributes
Service Provider
164
Client
162
service object

Fig. 8

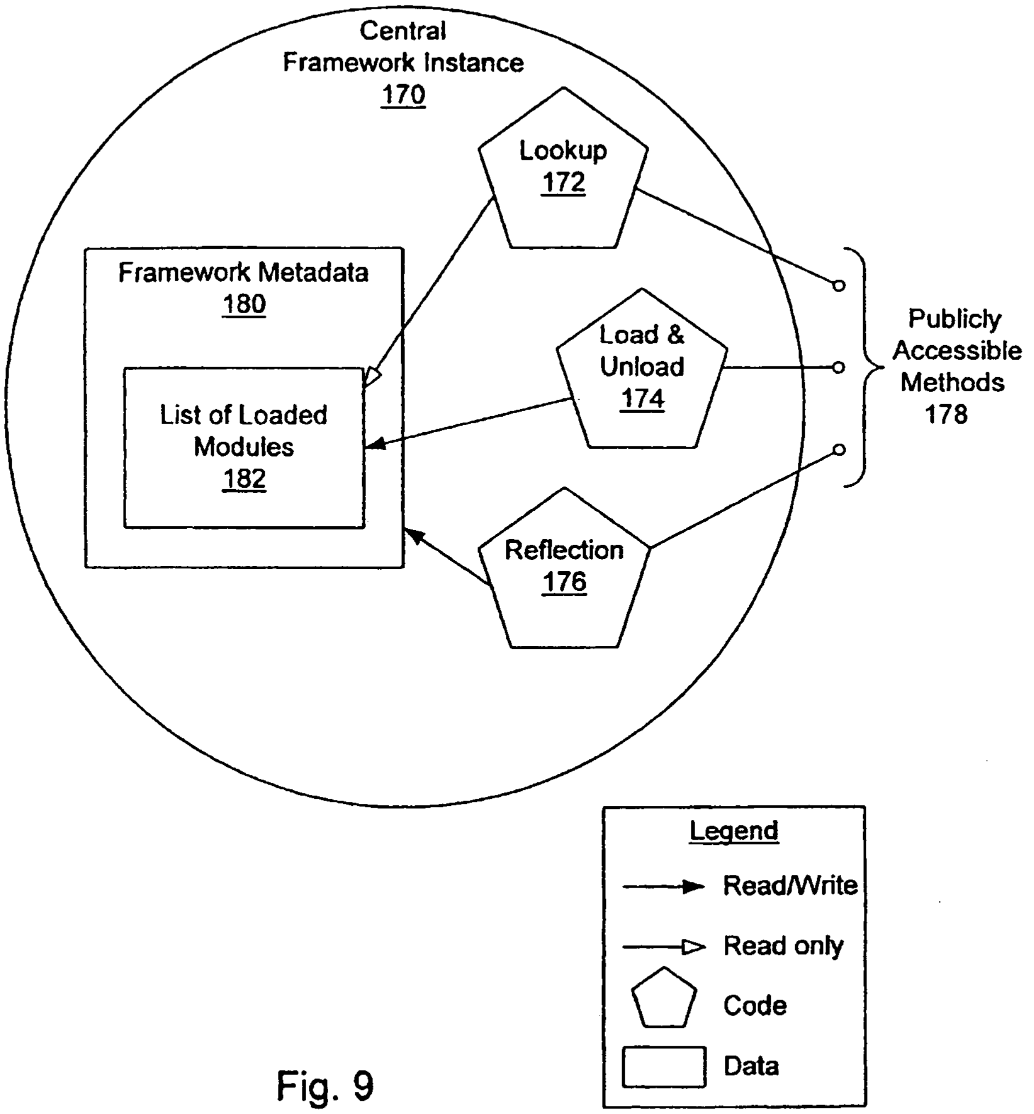
Central
Framework Instance
170
Lookup
172
Framework Metadata
180
Load &
Unload
174
List of Loaded
Modules
182
Publicly
Accessible
Methods
178
Reflection
176
Legend
Read/Write
Read only
Code
Data

Fig. 9

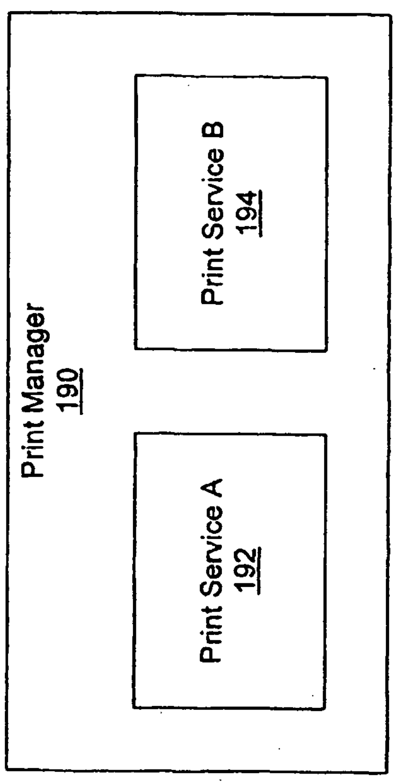
Print Manager
190
Print Service A
192
Print Service B
194

Fig. 10

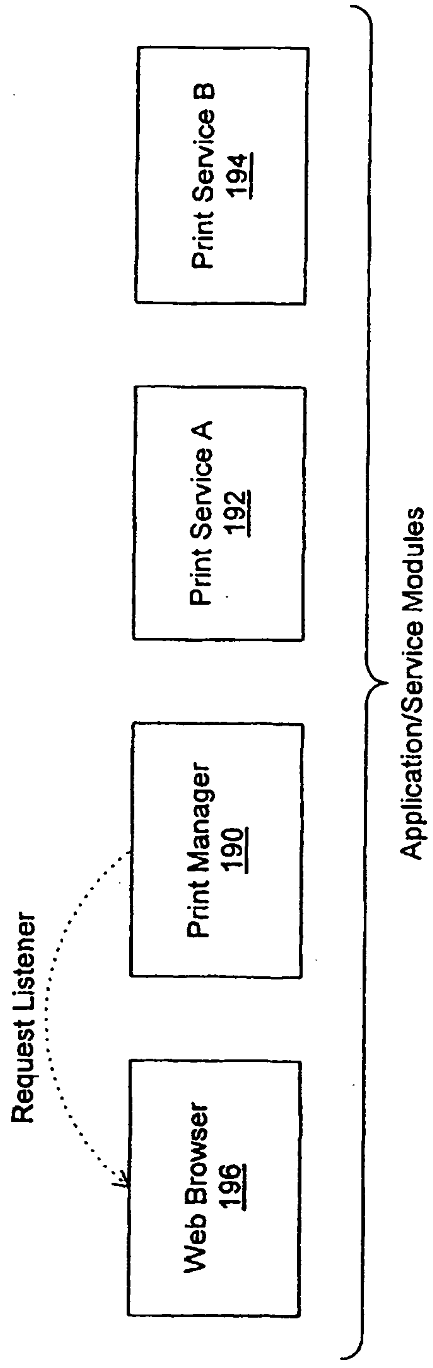
Web Browser
196
Request Listener
Print Manager
190
Print Service A
192
Print Service B
194
Application/Service Modules
198

Fig. 11

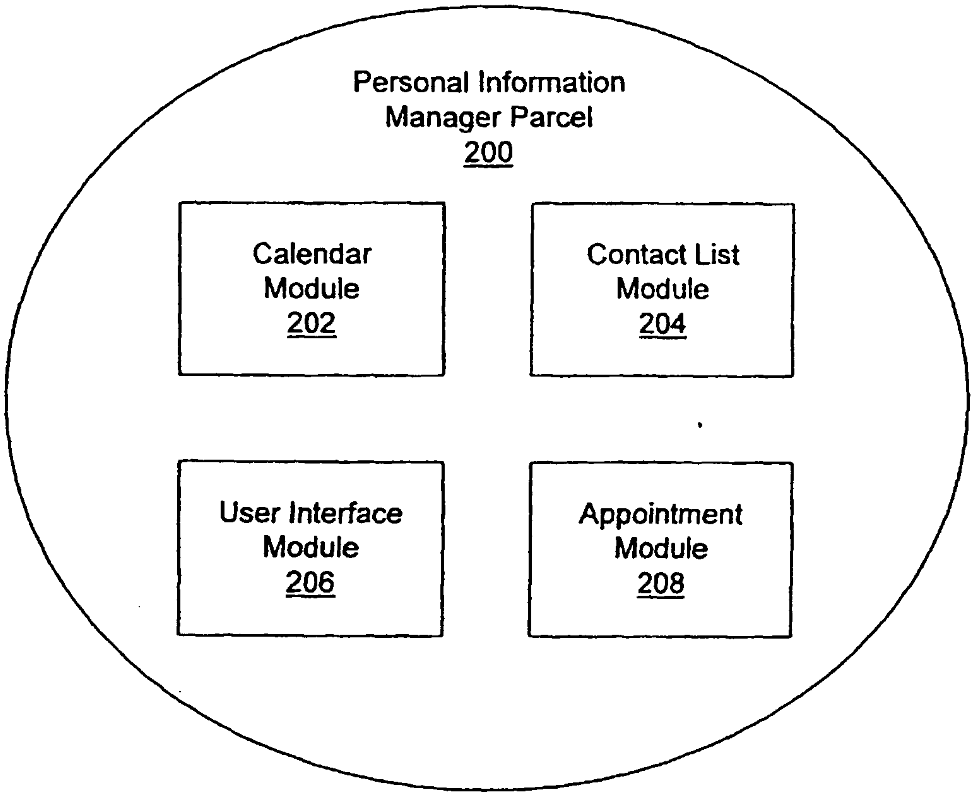
Personal Information
Manager Parcel
200
Calendar
Module
202
Contact List
Module
204
User Interface
Module
206
Appointment
Module
208

Fig. 12

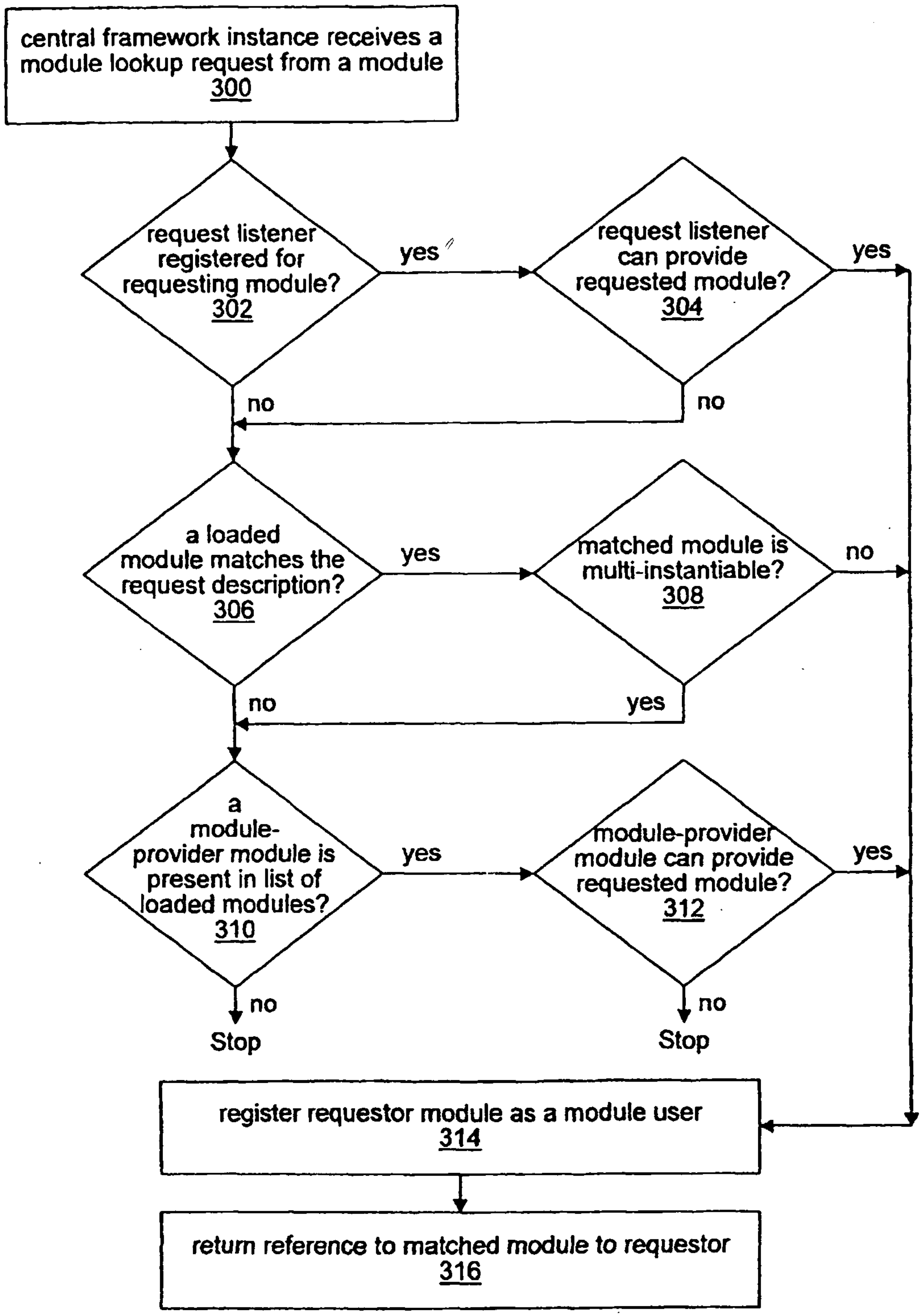
central framework instance receives a module lookup request from a module 300
request listener registered for requesting module? 302
yes
no
request listener can provide requested module? 304
yes
no
a loaded module matches the request description? 306
yes
no
matched module is multi-instantiable? 308
no
yes
a module-provider module is present in list of loaded modules? 310
yes
no
Stop
module-provider module can provide requested module? 312
yes
no
Stop
register requestor module as a module user 314
return reference to matched module to requestor 316

Fig. 13

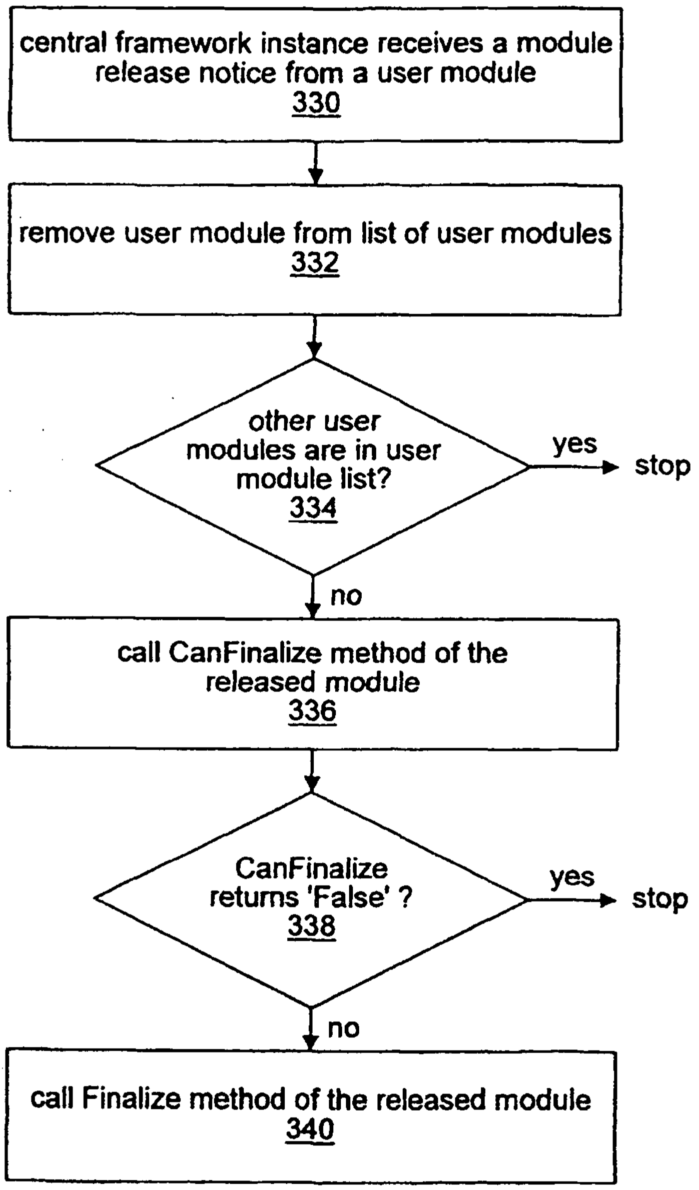
central framework instance receives a module release notice from a user module
330
remove user module from list of user modules
332
other user modules are in user module list?
334
yes
stop
no
call CanFinalize method of the released module
336
CanFinalize returns 'False' ?
338
yes
stop
no
call Finalize method of the released module
340

Fig. 14